**Project Requirements**

Building the Data Warehouse (Data Engineering)

Objective

Develop a modern data warehouse using SQL Server to consolidate sales data, enabling analytical reporting and informed decision-making.

Specifications

* Data Sources: Import data from two source systems (ERP and CRM) provided as CSV files.
* Data Quality: Cleanse and resolve data quality issues prior to analysis.
* Integration: Combine both sources into a single, user-friendly data model designed for analytical queries.
* Scope: Focus on the latest dataset only, historization of data is not required.
* Documentation: Provide clear documentation of the data model to support both business stakeholders and analytics teams.
* DDL(Data Definition Language).

Data management approach: Medallion Architecture

Design Layers:

**Bronze:** Raw data as-is from source

Obj: Traceability & Debugging

Obj types: Tables

Load Method:

* Full load
* Truncate and Insert

No transformation.

**Silver:** Clean & standardized data.

Obj: Prepare data for analysis.

Obj Types: Tables

Load:

* Full load
* Truncate and Insert

Transformation:

* Clean
* Normalization
* Enrichment
* Standardization,
* Derived columns.

**Gold:** Business-ready data

Obj: Provide data to be consumed for reporting & analytics.

Obj Type: Views.

Load:

* None

Transformation:

* Integration
* Aggregation
* Business Logic and rules,

Data Modeling:

* Star Schema
* Aggregated obj
* Flat tables.

General Principles

* Naming Conventions: Use snake\_case, with lowercase letters and underscores ( \_ ) to separate words.
* Language: Use English for all names
* Avoid Reserved Words: Do not use SQL reserved words as object names.

Table Naming Conventions

Bronze Rules

* All names must start with the source system name, and table names must match their original names without renaming.
* <sourcesystme>\_<entity>
  + <sourcesystem>: Name of the source system (e.g., crm, erp)
  + <entity> Exact table name from the source system.
  + Example crm\_customer\_info - Customer information from the CRM system.

Silver Rules

* All names must start with the source system name, and table names must match their original names without renaming.
* <sourcesystem>\_<entity>
  + <sourcesystem>: Name of the source system (e.g., crm, erp)
  + <entity> Exact table name from the source system.
  + Example crm\_customer\_info - Customer information from the CRM system.

Gold Rules

* All names must use meaningful, business-aligned names for tables, starting with the category prefix.
* <category>\_<entity>
  + <category>: Describes the role of the table, such as dim (dimension) or fact (fact table).
  + <entity>: Descriptive name of the table, aligned with the business domain (e.g., customers, products, sales)
  + Examples:
    - Dim\_customers - Dimension table for customer data
    - Fact\_sales - Fact table containing sales transactions.

Column Naming Conventions

Surrogate Keys

* All primary keys in dimension table must use the suffix \_key
* <table\_name>\_key
  + <table\_name>: Refers to the name of the table or entity the key belongs to.
  + \_key: A suffix indicating that this column is a surrogate key.
  + Example: customer\_key - Surrogate key in the dim\_customer table.

Technical Columns

* All technical columns must start with the prefix dwh\_, followed by a descriptive name indicating the column’s purpose.
* dwh\_<column\_name>
  + Dwh\_: Prefix exclusively for system-generated metadata
  + <column\_name>: Descriptive name indicating the column’s purpose
  + Example: dwh\_load\_date - System-generated column used to store the data when the record was loaded.

Stored Procedure

* All stored procedures used for loading data must follow the naming pattern:
* load\_<layer>
  + <layer>: Represents the layer being loaded, such as bronze, silver, or gold.
  + Example
    - Load\_bronze - Stored procedure for loading data into the bronze layer
    - Load\_silver - Stored procedure for loading data into the silver layer.

**Building the Bronze Layer**

* Analysing Source Systems: Interviewing source system experts
  + Business Context & Ownership
    - Who owns the data?
    - What Business Process it supports?
    - System & Data documentation
    - Data Model & Data Catalog
  + Architecture & Technology Stock
    - How is data stored? (SQL Server, Oracle, AWS, Azure…)
    - What are the integration capabilities? (API, Kafka, File Extract, Direct DB…)
  + Extract & Load
    - Incremental vs Full Loads?
    - Data Scope & Historical Needs
    - What is the expected size of the extracts?
    - Are there any data volume limitations?
    - How to avoid impacting the source system’s performance?
    - Authentication and authorization (tokens, SSH keys, VPN, IP whitelisting…)
* Coding: Data Ingestion
  + Bulk Insert
    - Truncate the tables before loading
    - Create a Procedure (bronze.load\_bronze)
    - Execute the Procedure
    - Add Print to make it more organized
    - Add TRY…CATCH: Ensures error handling, data integrity, and issue logging for easier debugging.
    - Track ETL Duration: Helps to identify bottlenecks, optimize performance, monitor trends, detect issues
* Validating: Data completeness & Schema Checks

**Building the Silver Layer**

* Analysing: Explore & Understand the Data
  + Details and Description of customers
  + Document & Visualize What You Understand from Data
* Coding: Data Cleaning
  + Check Quality of Bronze
  + CUST\_INFO
    - A primary key must be unique and not null
    - Using ROW\_NUMBER(): Assign a unique number to each row in a result set, based on a defined order.
    - Check for unwanted spaces in string values
    - Check the consistency of values in low cardinality column
    - Aim to store clear and meaningful values rather than using abbreviated terms (Apply UPPER() just in case mixed-case values appear later in your column.)
    - Re-run the quality check queries from the bronze layer to verify the quality of data in the silver layer.
    - Removed duplicates and Spaces
    - Handling Missing Data: Fills in the blanks by adding a default value
  + PRD\_INFO
    - Check for nulls or duplicate in Primary Key.
    - SUBSTRING(): Extracts a specific part of a string value from prd\_key to cat\_id
    - Replace the ‘-’, with ‘\_’, in the cat\_id to match with erp’s cat\_id using REPLACE()
    - Using the LEN(): Returns the number of characters in a string for the prd\_key
    - Check for Nulls and Negative numbers on prd\_cost
    - Use ISNULL(): Replaces NULL values with a specified replacement value
    - Prd\_line: Ask experts to Aim to store clear and meaningful values rather than using abbreviated terms
    - Check for invalid date order: End date must not be earlier than the Start date
    - For complex transformations in SQL, Narrow it down to a specific example and brainstorm multiple solution approaches. Solution: End Date = Start Date of the Next record -1.
    - LEAD(): Access values from the next row within a window.
    - CAST() AS DATE: to convert 00:00:00 to only date on both start and end date
  + CRM\_SALES
    - Check for Invalid Dates (Negative numbers or zeros can’t be cast to a date) or Check if the Length of date is equals to 8 for this scenario or check the boundaries with year
    - Using your CASE WHEN, convert the 0s and Length as NULL, Also CAST AS DATE by converting to VARCHAR first because you can’t convert INT to DATE
    - Apply same rules to all the DATES
    - Check if the Order Date is earlier than the shipping date or Due date
    - Check Sales, Quantity and Price Columns. The Sales equals to Quantity \* Price. And must not be negative, Zeros, Nulls for they are not allowed!
    - Incase of bad data:
      * Solution 1: Data Issues will be fixed direct in source system
      * Solution 2: Data Issues has to be fixed in Data Warehouse
    - If the sales is negative, zero, or null, derive it using Quantity and Price
      * If Price is zero or null, calculate it using Sales and Quantity
      * If Price is negative, convert it to a positive value.
      * Use ABS (Absolute) convert negative to positive.
    - Insert into silver table
  + ERP\_CUST\_AZ12
    - Transform the cid column to match the crm\_cst primary key
    - Check bdate (birthdate in the past and future)
    - Check the gender and make sure there is data standardization & consistency
    - Insert into silver layer
  + ERP\_LOC\_A101
    - Replace the ‘-’ in cid to match crm\_cst primary key
    - Insert into silver
  + ERP\_PX\_CAT\_GLV2
    - Check for unwanted spaces
    - Data Standardization & Consistency
    - Insert into silver
* Insert all into the Silver layer and then create a stored procedure
* Like Bronze layer, enhance the output of the Silver stored procedure
  + - Print messages for each section and step
    - Implement error handling
    - Print the duration of each step
    - Print the duration of loading Silver
* Keep to consistency: If you introduce an improvement, like better logging or error handling, in one stored procedure, apply it to the others to maintain consistent standards and benefits.
  + Write Data Transformation
  + Insert into Silver
* Validating: Data Correctness Checks
* Docs & Version: Data Documenting Versioning in GIT
  + Data Flow
  + Data Integration

METADATA COLUMS: Extra columns added by data engineers that do not originate from the source data.  
Example: Adding:

* create\_date: The record’s load timestamp.
* update\_date: The record’s last update timestamp.
* source\_system: The origin system of the record.
* file\_location: The file source of the record.

DERIVED COLUMNS: Create new columns based on calculations or transformations of existing ones.

DATA ENRICHMENT: Adding new, relevant data to enhance the dataset for analysis.

**Building Gold Layer**

* Analysing: Explore & Understand the Business Objects
* Coding: Data Integration
  + Build the Business Object
  + Choose Type Dimension vs Fact
  + Rename to friendly names
  + Create Dimension Customers
    - Not creating tables but using Views no stored procedure
    - Give the sql code an alias like “ci”
    - Always start your join with the master table using LEFT JOIN
    - Join all the customer tables
    - After joining table, check if any duplicates were introduced by the join logic. By using a GROUP BY function.
    - Data integration of the gender columns
    - Follow the naming convention, rename the columns
    - Sort the columns into logical groups to improve readability
    - Generate a Surrogate Key
    - Create the Gold dimension customer VIEW
  + Create the dimension for Product
    - Filter out all historical data and leave the current data that ends with NULL
    - Use the GROUP BY function to check for duplicates
    - Sort the columns into logical groups to improve readability
    - Rename the columns
    - Create the gold dimension product VIEW
  + Create the Fact Sales
    - Use the dimension’s surrogate keys instead of IDs to easily connect facts with dimensions
    - Join the sales details with the dimension products
    - Sort the columns into logical groups to improve readability
    - CREATE VIEW gold.fact\_sales
    - Quality checks
    - Check if all dimension tables can successfully join to the fact table
    - Build the Star Schema Model
      * Draw Customers and products tables with all Columns
      * In a Star Schema, the relationship between fact and dimensions is 1-to-many (1:N). 1 dimension and many in facts.
* Validating: Data Integration Checks
* Docs & Version: Documenting and Versioning in GIT
  + Data Model
    - Build the Star Schema Model
      * Draw Customers and products tables with all Columns
      * In a Star Schema, the relationship between fact and dimensions is 1-to-many (1:N). 1 dimension and many in facts.
  + Data Catalog
    - Done on GIThub
  + Data Flow
  + Commit Code in Git Repo

DATA MODELING:

Types

Conceptual Data Model: Only on entity focus in the table (BIG PICTURE)

Logical Data Model: Specify columns and relationship in the table(BLUE PRINT)

Physical Data Model: Adding all little details(IMPLEMENTATION Databricks can do this)

STAR SCHEMA vs SNOWFLAKE SCHEMA

Star Schema: Has Fact table in the middle surrounded by the Dimensions table (Big Dimension)

Snowflake Schema: Has Facts table in the middle surrounded by Dimensions and further broken down Dimensions table

DIMENSION vs FACTS

Dimension: Contains descriptive information that give context to your data. They answer the questions of “Who?, What?, Where?”

Facts: Quantitative information that represents events. Answers the question “How much?, How many?”

SURROGATE KEY

System generated unique identifier assigned to each record in a table.

How to generate?; DDL based generation, Query based using Window function (ROW\_NUMBER)

| Column | Data Type | Description |
| --- | --- | --- |
| customer\_key | INT | Surrogate key uniquely identifying each customer record in the dimension table |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |